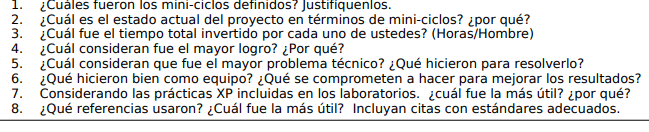
**CICLO #1**

**SILKROAD**

**MIGUEL SANDOVAL**

**SANTIAGO GOMEZ**

**RETROSPECTIVA**



* Mini ciclo 1: agregar las nuevas clases al proyecto ya creado en shapes, las cuales fueron silkRoad, Robot y Store ya que es la integración de lo que se quiere lograr en este proyecto nuevo.
* Mini ciclo 2: ayudarnos de las clases de shapes para la implementación de figuras básicas y así poder representar las tiendas y los robots.

1. El estado actual es un buen funcionamiento del juego silkRoad con algunos errores técnicos, también contamos con la parte en astah, pero no sabemos si este bien estructurado
2. 6 horas: miguel Sandoval, 7 horas: Santiago Gómez
3. El mayor logro fue integrar las nuevas clases junto con el proyecto shapes, creando un sistema funcional y entendible.
4. El mayor problema fue implementar el código y que se pudiera compilar junto con las otras clases ya que todavía hay errores de funcionamiento frente a la clase canvas. Aun no lo resolvemos por completo.
5. Organizar el trabajo en partes, primero miramos lo que necesitábamos implementar junto con los mini ciclos y una buena comunicación al momento de trabajar en el blue j.
6. La de programación en parejas, esta nos ayudó a minimizar el margen de error en el código.
7. Documentación oficial de BlueJ: <https://www.bluej.org>

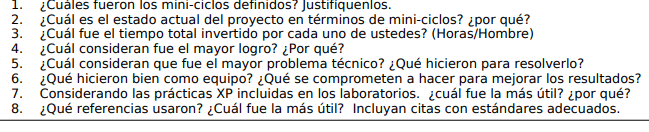
**CICLO #2**

**SILKROAD**

**MIGUEL SANDOVAL**

**SANTIAGO GOMEZ**

**RETROSPECTIVA**



1. Decidimos dividir el trabajo en 3 partes principales:

**Mini-ciclo 1: Mejorar cómo crear la Ruta de la Seda (Construcción Avanzada (Requisito 10))**

* Hacer que se pudiera crear usando el formato que pide el problema de la maratón
* También permitir crearla pasando arrays con la información de tiendas y robots
* Esto era importante porque facilitaba probar nuestro código con diferentes casos

**Mini-ciclo 2: Hacer los robots más inteligentes (Movimientos Inteligentes (Requisito 11))**

* Los robots ahora pueden decidir por sí solos hacia dónde moverse para ganar más dinero
* Agregamos un método para mover todos los robots de una vez
* Fue la parte más complicada porque teníamos que hacer que eligieran bien

**Mini-ciclo 3: Llevar estadísticas (Sistema de Estadísticas (Requisitos 12 y 13))**

* Contar cuántas veces se vacía cada tienda
* Guardar cuánto dinero gana cada robot en cada movimiento
* Permitir consultar toda esta información cuando se necesite

1. El estado actual del ciclo 2, se completaron los métodos de los tres ciclos, no tiran errores ni fallos al momento de probarlo en C2Test, es decir tienen buena funcionalidad, pero no logramos arreglar el tema visual al ejecutar el juego desde la clase silkroad directamente, tiene algunos errores, pero de igual manera si se realizan los métodos los hace.
2. 12 horas: miguel Sandoval, 12 horas: Santiago Gómez
3. El mayor logro fue integrar los nuevos métodos del ciclo2, que todo funcionara de manera correcta sin errores ni fallos.
4. El mayor problema fue dejar la parte visual funcionando de la mejor manera ya que al crear robot se crean otros sin uno pedirlo, es algo que trataremos de arreglar para el ciclo3, y que tanto la parte visual como al probar los métodos del C2Test funcione de manera correcta.
5. Lograr resolver dudas que el otro compañero tenia y así ir solucionado todo de la mejor manera, también ayudándonos mutuamente para la parte del código y que no tuviera errores ni fallos, y lográramos una buena implementación.
6. La de programación en parejas, esta nos ayudo a minimizar el margen de error en el código.

* Documentación oficial de BlueJ: <https://www.bluej.org>
* Curso de Moodle de DOPO: <https://campusvirtual.escuelaing.edu.co/moodle/course/view.php?id=95&section=4>
* <https://www.ibm.com/docs/es/spss-modeler/saas?topic=language-object-oriented-programming>
* Problema j: <https://drive.google.com/file/d/1jlwi4nlcb0RpT7L4wC8rCcdpl5EZaDgR/view>
* Video Problema j: <https://www.youtube.com/watch?v=mcwVytY5ooU>

**CICLO #3**

**SILKROAD**

**MIGUEL SANDOVAL**

**SANTIAGO GOMEZ**

**RETROSPECTIVA**

Texto

El contenido generado por IA puede ser incorrecto.

1. Decidimos dividir el trabajo en 3 partes principales:

**Mini-ciclo 1: Implementación del algoritmo de solución (Requisito 14 - solve)**

* Implementar la clase SilkRoadContest separada de SilkRoad para mantener responsabilidades claras
* Desarrollar el algoritmo de programación dinámica con bitmask para calcular la ganancia máxima diaria
* Implementar el algoritmo TSP (Traveling Salesman Problem) para permitir que los robots visiten múltiples tiendas en un día
* Fue la parte más compleja porque requería entender el problema de la maratón y diseñar un algoritmo óptimo

**Mini-ciclo 2: Simulación visual de la solución (Requisito 15 - simulate)**

* Crear el método simulate() que use la clase SilkRoad para mostrar visualmente la solución
* Integrar SilkRoadContest con SilkRoad de manera que el simulador muestre paso a paso cada día
* Asegurar que la simulación sea clara y fácil de seguir para el usuario

**Mini-ciclo 3: Pruebas exhaustivas y refinamiento**

* Crear SilkRoadContestTest con pruebas unitarias usando JUnit
* Crear SilkRoadContestCTest con casos de prueba comunes del problema
* Actualizar SilkRoadAcceptanceTest para incluir las pruebas visuales del Ciclo 3
* Corregir errores en las pruebas de ciclos anteriores que surgieron durante la integración

1. El estado actual del Ciclo 3 está completo y funcional. Se implementaron exitosamente ambos requisitos: El método solve() calcula correctamente la ganancia máxima diaria usando un algoritmo robusto de DP, el método simulate() muestra visualmente la solución paso a paso, todas las pruebas unitarias pasan sin errores ni fallo. La separación entre SilkRoad (simulador) y SilkRoadContest (resolvedor) quedó clara y bien definida
2. 15 horas: miguel Sandoval, 15 horas: Santiago Gómez
3. El mayor logro fue implementar exitosamente un algoritmo complejo de programación dinámica que resuelve el problema de la maratón ICPC 2024. Específicamente: Diseñar e implementar un DP con bitmask que asigna óptimamente robots a subconjuntos de tiendas, integrar un algoritmo TSP que permite a cada robot visitar múltiples tiendas en secuencia, lograr que el algoritmo pase todos los casos de prueba, incluyendo el ejemplo oficial con resultado esperado [0, 10, 35, 50, 50, 60], mantener una arquitectura limpia donde SilkRoad solo simula y SilkRoadContest resuelve el problema
4. El mayor problema fue **entender correctamente la naturaleza del problema de la maratón**. Inicialmente interpretamos mal que: Los robots podían visitar solo una tienda por día (interpretación incorrecta). La realidad es que cada robot puede visitar múltiples tiendas en secuencia, lo que hace el problema significativamente más complejo
5. Logramos una excelente coordinación en este ciclo: Revisión conjunta del problema de la maratón para asegurar una comprensión correcta, comunicación constante para resolver dudas sobre el algoritmo de DP Apoyo mutuo en la comprensión de conceptos avanzados (bitmask, TSP, programación dinámica)
6. La de programación en parejas, esta nos ayudó a minimizar el margen de error en el código.

* Documentación oficial de BlueJ: <https://www.bluej.org>
* Curso de Moodle de DOPO: <https://campusvirtual.escuelaing.edu.co/moodle/course/view.php?id=95&section=4>
* <https://www.ibm.com/docs/es/spss-modeler/saas?topic=language-object-oriented-programming>
* Problema j: <https://drive.google.com/file/d/1jlwi4nlcb0RpT7L4wC8rCcdpl5EZaDgR/view>
* Video Problema j: <https://www.youtube.com/watch?v=mcwVytY5ooU>

**CICLO #4**

**SILKROAD**

**MIGUEL SANDOVAL**

**SANTIAGO GOMEZ**

**RETROSPECTIVA**

Texto

El contenido generado por IA puede ser incorrecto.

* 1. Decidimos dividir el trabajo en 3 partes principales:
* **Mini-ciclo1**: Refactorización de Paquetes - Para este ciclo decidimos organizar el trabajo en tres mini-ciclos que nos permitieran avanzar de manera ordenada. El primero fue la refactorización de paquetes, donde separamos todo el código en tres paquetes: shapes, test y silkRoad. Esto lo hicimos porque el código estaba mezclado y necesitábamos que fuera más modular y reutilizable. Tuvimos que cambiar protected a public en algunos métodos y agregar imports correctamente para que todo comunicara entre paquetes.
* **Mini-ciclo2**: **Extensión del Sistema** - En el segundo mini-ciclo nos enfocamos en la extensión del sistema, implementando cuatro tipos diferentes de tiendas y cuatro tipos de robots, cada uno con su propio comportamiento. Esto fue importante porque demostraba que nuestro diseño era verdaderamente extensible, es decir, que podíamos agregar nuevas cosas sin romper lo que ya estaba hecho. Implementamos clases y métodos abstractos para que cada tipo pudiera tener su propia lógica.
* **Mini-ciclo3**: Pruebas y Validación - Finalmente, el tercer mini-ciclo estuvo dedicado a las pruebas, donde creamos 20 pruebas unitarias, pruebas de aceptación para asegurar que todo funcionaba como esperábamos y que los usuarios podían interactuar con el sistema sin problemas. Escribimos pruebas para cada tipo de tienda, cada tipo de robot, y todas sus interacciones posibles.

1. El ciclo4 está completamente terminado y funcional en todos sus mini-ciclos. El primer mini-ciclo de refactorización fue exitoso porque logramos separar el código en dos paquetes sin que nada se rompiera, todos los imports están bien configurados y las clases de un paquete pueden acceder a las del otro sin problemas. En el segundo mini-ciclo implementamos exitosamente los ocho nuevos tipos (tiendas y robots) y lo más importante es que la clase SilkRoad no necesitó casi ningún cambio, lo que comprueba que el sistema es extensible. El tercer mini-ciclo también está completo con todas las pruebas funcionando correctamente, demostrando que cada tipo de tienda y robot se comporta como se esperaba. Todo funciona porque desde el inicio pensamos bien en la arquitectura y usamos conceptos como herencia y polimorfismo de manera correcta.
2. 30 horas: Santiago Gómez, 30 horas Miguel Sandoval
3. El mayor logro fue demostrar que realmente entendimos cómo funcionan la herencia y el polimorfismo en Java. Logramos crear ocho tipos diferentes de entidades (cuatro tiendas y cuatro robots) que se comportan completamente diferente una de la otra, pero todas funcionan juntas sin necesidad de modificar la clase SilkRoad. Por ejemplo, la tienda Fighter rechaza robots pobres, la tienda Generous duplica el dinero, el robot Tender solo toma la mitad del dinero, y el robot NeverBack no puede volver a su posición inicial. Aunque son comportamientos muy diferentes, el sistema los maneja de manera elegante gracias a que usamos métodos abstractos y polimorfismo. Lo que nos hace especialmente orgullosos es que nuestro código de ciclos anteriores sigue funcionando exactamente igual, lo que demuestra que no rompimos nada mientras extendíamos el sistema. Esto es exactamente lo que pide la materia: crear código que pueda crecer sin necesidad de cambiar lo que ya existe.
4. El mayor problema que enfrentamos fue un error de acceso entre paquetes. Cuando intentamos compilar, nos salía un mensaje diciendo que setposition()  tenía acceso protected en Rectangle y Circle, pero necesitábamos llamarlo desde la clase Store que estaba en otro paquete. Al principio pensamos que era un error de diseño, pero después nos dimos cuenta de que simplemente necesitábamos cambiar protected a public. Esto fue importante porque aprendimos que los modificadores de acceso deben adaptarse al contexto: en ciclos anteriores teníamos todo en un solo paquete, así que protected tenía sentido, pero ahora que teníamos tres paquetes, necesitábamos que fuera público. No fue un gran cambio técnico, pero fue valioso aprender que el mismo código puede necesitar ajustes menores cuando cambia la estructura del proyecto.
5. Hicimos bien mantener el código simple y organizado desde el principio, sin complicarlo más de lo necesario. Nos comunicábamos constantemente, probábamos cada cosa conforme la hacíamos en lugar de dejar todo para el final, y documentamos el código al mismo tiempo que lo escribíamos. Para mejorar en próximos ciclos, nos comprometemos a escribir las pruebas antes del código y a revisar mejor el trabajo del otro antes de integrarlo al proyecto.
6. Las dos prácticas más útiles fueron Simplicity y Refactoring. La simplicidad nos ayudó a diseñar el sistema de una forma que fuera fácil de entender y de extender. No inventamos cosas complicadas, simplemente usamos métodos abstractos que cada tipo implementaba de su propia manera. El refactoring fue importante porque cuando encontrábamos que algo no estaba bien (como el problema de protected a public), lo arreglábamos sin miedo de romper lo demás. Esto nos dio confianza para mejorar el código sobre la marcha. La programación en parejas también fue útil cuando implementábamos tipos nuevos complejos, porque uno revisaba lo del otro en tiempo real. Estas prácticas juntas nos permitieron crear un código que funciona bien y que es fácil de cambiar si es necesario.

* Documentación oficial de BlueJ: <https://www.bluej.org>
* Curso de Moodle de DOPO:

<https://campusvirtual.escuelaing.edu.co/moodle/course/view.php?id=95&section=4>

* <https://www.ibm.com/docs/es/spss-modeler/saas?topic=language-object-oriented-programming>
* Problema j: <https://drive.google.com/file/d/1jlwi4nlcb0RpT7L4wC8rCcdpl5EZaDgR/view>
* Video Problema j: <https://www.youtube.com/watch?v=mcwVytY5ooU>